ISSN 1330-3651
UDC/UDK 004.412:004.43 Python

ESTIMATING COMPLEXITY OF PROGRAMS IN PYTHON LANGUAGE

Sanjay Misra, Ferid Cafer

Original scientific paper

In this paper, a complexity metric for Python language is formulated. Since Python is an object oriented language, the present metric is capable to evaluate any object-oriented language. We validate our metric with case study, comparative study and empirical validation. The case study is in Python, Java and C++ and the results prove that Python is better than other object-oriented languages. Later, we validate the metric empirically with a real project, which is developed in Python.

Keywords: *complexity metric, Python, software complexity, software development*

Procjena složenosti programa u Python jeziku

Izvorni znanstveni članak

U ovom radu formulirana je metrička složenost za jezik Python. Budući je Python objektno orijentiran jezik, postojeća metričnost je u stanju procijeniti bilo koji objektno usmjeren jezik. Potvrđujemo našu metričnost studijom slučaja, usporednom studijom i empirijskom provjerom valjanosti. Studija slučaja je u Python, Java i C ++ jeziku, a rezultati pokazuju da je Python bolji od ostalih objektno orijentiranih jezika. Kasnije smo provjerili metričnost empirijski sa stvarnim projektom, koji je razvijen u Python jeziku.

Ključne riječi: *metrička složenost, Python, složenost softvera, razvoj softvera*

1
Introduction
Uvod

Software metrics determine the degree of maintainability of software products, which is one of the important factors that affect the quality of any kind of software. Furthermore, software metrics provide useful feedback to the designers to impact the decisions that are made during design, coding, architecture, or specification phases. Without such feedback, many decisions are made in ad hoc manner.

Fenton defines measurement as the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined unambiguous rules [6]. In software engineering software metrics are the only tools to control the quality of software. Furthermore, requirement to improve the software quality is the prime objective, which promoted the research projects on software metrics technology. It is always hard to control the quality if the code is complex. Complex codes always create problems to software community. It is hard to review, test, maintain as well as manage such codes. As a consequence, those handicaps increase the maintenance cost and the cost of the product. Due to these reasons, it is strongly recommended to control the complexity of the code from the beginning of the software development process. Software metrics help to achieve this goal.

In fact, there are no strict rules, which can prove that code is good or bad. Several researchers proposed different criteria for developing the good and quality software. Although they consider the different aspect of the quality one thing is common in all the criteria i.e. the code should be agile [31]. Specially, in object oriented software developments, which have an extraordinary acceptance in last two decades, it is a must. In other words, the code should be written in such a way that it should be reused and adopted. If code is not agile then it is hard to replace (add or remove) modules and it will waste the time of software professionals. It also affects the maintainability aspects.

In the past, researchers proposed their methodologies for evaluating codes, which were written in procedural languages [12], such as C. Later, studies focused on object-oriented (OO) programming languages, e.g. C++ [1, 2, 3] and Java [35, 36]. Software metrics for other languages and technologies such as XML and Web services [13, 14, and 24] can also be found in the literature. However, to evaluate codes written in Python language has not found proper attention as we were expecting. In fact, today, Python is not as popular as Java and C++ but due to its unique features, it is a more comfortable language for software development. It is gaining popularity day by day in the software community. Several conferences and workshops devoted to Python including SciPy 2009 (India) [16], RuPy '09 (Poland) [17], FRUncon '09 (USA) [18], ConFoo.ca 2010 (Canada) [19], are proving the importance of Python. It is not the end of the success stories of Python. It is highly understandable in comparison to other OO languages, hence less expensive to maintain. Famous enterprises like Google and YouTube chose to use Python.

One way to evaluate the complexity of the Python code is through the metrics developed for procedural languages. However, all the available metrics cover only specific features of the language. For example, if we apply line of code then only size is covered, similarly McCabe's Complexity metrics only cover the control flow of the program. In general, the metrics applicable to the procedural languages do not fit to the modern OO languages such as C++ [3]. Additionally, most of the available metrics and models for OO languages are applicable when the object-oriented code is complete or about to complete. Therefore, they provide information too late to help in improving the quality of the code. Object oriented approach requires significant efforts early in development life cycle to identify objects, classes, attributes operation and their relationship. Encapsulations, inheritance and polymorphism require the designer to carefully structure the

design in early phases [32]. Therefore, the available metrics, which are applicable at the completion of code, are not applicable in OO environment.

By keeping all these issues in mind, in the present work we are evaluating the complexity of Python code by identifying all the factors, which may play important roles in the complexity of the code. It is important to note that Python is an OO language, it includes most of the features of other OO languages; however, differences occur in the main program body. In addition to that, execution and dynamic typing provide big productivity gains over Java; Python programs need less extraneous endeavor (i.e. cleaner code) [21]. In this respect Python is a very useful and important language and the proper metrics for this language are still not developed. This factor motivates us to work on it and produce such a metric, which can cover all the features of Python as well as cover all the aspects of complexity. Consequently, a new approach, which is unification of all the attributes, is presented. As the preliminary work of this study, the metric is presented in ICCSA 2010 [33]. In the present work, we extend our conference presentation and validate our work through empirical validation, which includes case study, comparative study and applicability on a real project. Our case study is evaluated in three different object oriented languages. Furthermore, we apply our metric on a real project to prove its real applicability and usefulness.

The paper is organized in the following way. We discuss the importance of Python language and the available metrics for this language in Section 2. Following this, we propose our metric in Section 3. We demonstrate the metric with a programming example in section 4. We empirically validate our metric through case study, comparative study and through a real project in section 5. The conclusion drawn on this work is in Section 6.

2
The literature survey
Pregled literature

Python is a programming language that lets the programmer to work more quickly to integrate systems more effectively [10, 11]. It is a free of charge language for commercial purpose. It runs on all major operating systems including Windows, Linux/Unix, Mac OS X, and also has been ported to the Java and .NET virtual machines [10]. Besides those features, Python is an effective language especially for the software development for embedded systems. It can also be used in web development [4, 5]. It is an ideal language to solve problems, especially on Linux and UNIX, for building software applications in life science research and development, and processing in natural languages [7-9].

In case of embedded system, where inexpensive components and maintenance are demanded, Python may provide best solution. With Python, one can achieve these goals in terms of small size, high reliability and low power consumption. In addition, the developers who have background of Java, C and/or Visual Basic [4] can learn it without major effort. In fact, it offers features of mixture of programming languages. It provides most of the features of OO languages in a powerful and simple way.

In addition to these powerful features of this language, no proper techniques are available to evaluate the quality of Python code to our knowledge. We could not find a single metric in a published form except some online articles [25-30]. These available articles are related to the available tools, which are limited to calculate the simple metrics. For example, Pythius [26] tool calculates the complexity of Python Code by computing simple metrics such as ratio of comments to code lines, module and function size.

Another tool 'snakefood' proposed by Martin Blais [27] provides the dependency graphs for Python. It can be useful to calculate McCabe's Cyclomatic complexity for Python code. Pygenie tool developed by David Stanek [28] also calculates the McCabe's Cyclomatic complexity for Python code. Reg Charney has also developed an open source code complexity measurement tool named as PyMetrics [29] which is capable of counting the following metrics: block count, maximum block depth, number of doc strings for Python classes, number of classes, number of comments, number of inline comments, total number of doc strings, number of function doc strings, number of functions, number of keywords used, number of lines, number of characters and number of multiple exit functions. Another tool available online is Pyntch [30]. Pinch (PYthoN Type CHecker) is a static code analyzer which detects possible run time errors before actually running a code.

All of the above tools are effective in evaluating the quality of the Python language only up to an extent. Most of them are confined to compute simple metrics, which give only an idea for some specific attributes; none of them are capable to evaluate majority of attributes in a single metric.

3
Proposed metric
Predložena metričnost

If we analyse object-oriented software, we will find that software consists of several classes with a main program. Accordingly, complexity of the object-oriented code depends on the complexity of the class and the complexity of main program. It is common observation that most of the available metrics do not care for the complexity of the main program in object-oriented systems. In our proposal, we consider all the factors, which are responsible for increasing complexity of the class as well as main program. Actually main program is the main component, which differentiates Python with other object oriented languages. We will prove our claims in empirical validation, section 5. Further, complexity of a system depends on the following factors:

1. Complexity due to classes. Class is a basic unit of object oriented software development. All the functions are distributed in different classes. Further classes in the object-oriented code either are in inheritance hierarchy or distinctly distributed. Accordingly, the complexity of all the classes is due to classes in inheritance hierarchy and the complexity of distinct classes.
2. Complexity due to global factors: The second important factor, which is normally neglected in calculating complexity of object-oriented codes, is the complexity of global factors in main program.
3. Complexity due to coupling: Coupling is one of the important factors for increasing complexity of object-oriented code.

The other factors like cohesion and methods are considered the complexity factors inside the class.

Accordingly, we propose that the Complexity of the Python code is defined as:

Software Metric for Python (SMPy)
= CIclass + CDclass+ Cglobal + Ccoupling, (1)

CIclass = Complexity due to Inheritance
CDclass = Complexity of Distinct Class
Cglobal= Global Complexity
Ccoupling= Complexity due to coupling between classes.

Before calculating the complexity of Inheritance and distinct classes, we will estimate the complexity of a class, which will later become the part of inheritance hierarchy or of distinct classes. Accordingly, we will first compute the complexity of a simple class named as Cclass.

Cclass can be defined as:

Cclass = weight (attributes) + weight (variables) + weight (structures) + weight (objects) – weight (cohesion) (2)

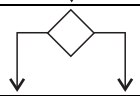
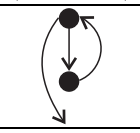
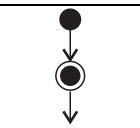
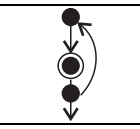
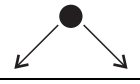
where, weight of attributes W (attribute) is defined as:

W (attributes) = 4*AND + MND, (2.1)

where, AND = Number of Arbitrarily Named Distinct Variables/Attributes
MND = Number of Meaningfully Named Distinct Variables/Attributes.
Weight of variable W(variables) is defined as:

W(variables) = 4*AND+MND. (2.2)

Table 1 *Values of structures*
Tablica 1. *Vrijednosti struktura*

Category	Value	Flow Graph
Sequence	1	
Condition	2	
Loop	3	
Nested Loop	3	-
Function	2	
Recursion	3	
Exception	2	

Arbitrary and Meaningful Variables and attributes are one of the causes of complexity. Further, if a variable's name is arbitrarily given, then the comprehensibility of that code will be lower [20]. Therefore, variables and attributes of classes should have meaningful names. Although, it is suggested that the name of the variables should be chosen in such a way that it is meaningful in programming, there are developers who do not follow this advice strictly. If the variable names are taken arbitrarily, it may not be a grave problem if the developer himself evaluates the code. However, it is not the case in real life. After the system is developed, especially during maintenance time, arbitrarily named variables may increase the difficulty in understanding four times more [20] than the meaningful names.

Weight of structure W(structures) is defined as:
W(structures) = W(BCS), (2.3)
where, BCS are basic control structure.

Structures/basic Control Structures (BCS's): Sequences' complexity depends on its mathematical expression [15]. Functions help tidiness of a code. Also, they may increase the reusability of the code. However, each function call disturbs the fluency of reading a code. Loops are used to repeat a statement for more than one time, although, they decrease computing performance. Especially, the more nested loops there are, the more time it takes to run the code. In addition, human brain has similarities with computers in interpreting a data [22, 23]. Conditional statements are used to make a program dynamic. On the other hand, by presenting more combinations they decrease the easiness of grasping the integrity of a program. Therefore conditional statements can be thought to be sequences built up in different possible situations. If the conditional situations are nested, then the complexity becomes much higher. The situation in exceptions is similar. We are assigning the weights for each basic control structure followed by the similar approach of Wang [15]. Wang proved and assigned the weights for sub conscious function, meta cognitive function and higher cognitive function as 1, 2 and 3 respectively. Although we followed the similar approach with Wang, we made some modifications in the weights of some Basic Control Structures as shown in Tab. 1.

Weight of objects W (object) is defined as:
W(object) = 2. (2.4)

Creating an object is counted as 2, because while creating a function constructor is automatically called. Therefore, it is the same as calling a function or creating an object. Here it is meant to be the objects created inside a class.

Weight of cohesion is defined as:
W(cohesion) = MA/AM, (2.5)

where, MA = Number of methods where attributes are used
AM = Number of attributes used inside methods.
While counting the number of attributes there is no any importance of AND or MND.

Notes:
Function call: during inheritance, calling super class's constructor is not counted.
Global variable: static attribute is counted as a global variable.

Cglobal can be defined as:
Cglobal=W (variables+structures+objects). (3)

Weight of variable W(variable) is defined as:
W(variables) = 4*AND+MND. (3.1)
The variables are defined globally.

Weight of structure W(structure) is defined as:
W(structures) = W(BCS)+obj.method. (3.2)

where, BCS are basic control structure, and those structures are used globally. 'obj.method' calls a reachable method of a class using an object. 'obj.method' is counted as two, because it calls a function written by the programmer.

Weight of objects W(object) is defined as:
W(objects) = 2. (3.3)

Creating an object is counted as 2, as it is described above. Here it is meant to be the objects created globally or inside any function which is not a part of any class.

Notes: Exception: while calculating try catch statement, only the numbers of "catch"es are counted as 2. "try" itself is not counted.

CIclass can be defined as:

There are two cases for calculating the complexity of the Inheritance classes depending on the architecture:

- If the classes are in the same level then their weights are added.
- If they are children of a class then their weights are multiplied due to inheritance property.

If there are *m* levels of depth in the OO code and level *j* has *n* classes then the complexity of the system due to inheritance is given as:

$$\text{CIclasses} = \prod_{j=1}^{m}\left[\sum_{k=1}^{n}\text{Cclass}_{jk}\right] \quad (4)$$

CDclass can be defined as:

$$\text{CDclass} = \text{Cclass}(x) + \text{Class}(y) + \ldots \quad (5)$$

Note: all classes, which are neither inherited nor derived from another, are part of CDclass even if they have caused coupling together with other classes.

Coupling is defined as:

$$\text{Coupling} = 2^{c} \quad (6)$$

c = Number of connections.

A method, which calls another method in another class, creates coupling. However, if a method of a class calls the method of its super class, then it is not considered to be coupling. In order to provide a connection there have to be two entities. It means one connection is in between two points. Thus, the base number is taken as 2. Another reason for that is function call has a value of 2; c is total number of connection made from one method to other method(s) in another class. It is taken as 'to the power', because each connection makes a significant increment in cognitive comprehensibility in software.

4
Demonstration of the metric
Demonstracija metričnosti

We have demonstrated our proposed complexity metric given by equation (1) by a programming example written in Python language. The class diagram for this programming example is given in Fig. 1. The complete code for this program is given in the Appendix A. Further, the computations of the weights of basic control structures are also demonstrated in Appendix B (Tab. 10).

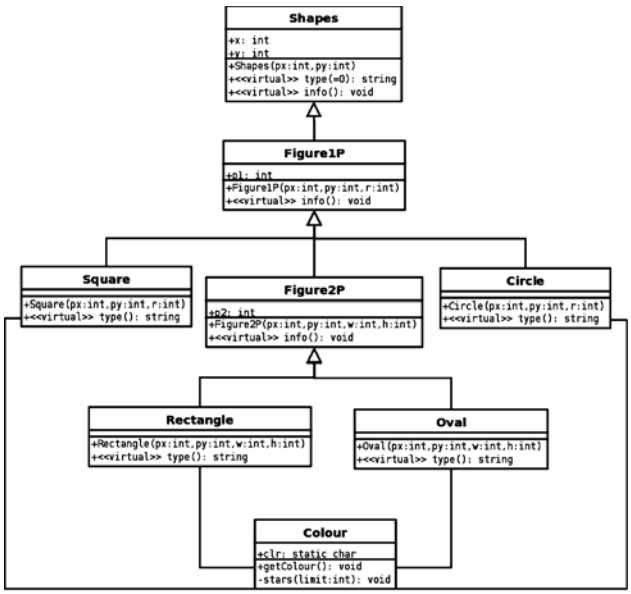


Figure 1 *Shapes – Class diagram*
Slika 1. *Oblici –Dijagram razreda*

Table 2 *Class complexity*
Tablica 2. *Složenost razreda*

class	attribute	Structures	variable	object	MA	AM	cohesion	Cclass
Colour	0	19	2	0	0	0	0	21
Shapes	2	6	0	0	2	2	1	7
Figure1P	1	8	0	0	2	1	2	7
Square	0	27	0	2	0	0	0	29
Circle	0	27	0	2	0	0	0	29
Figure2P	1	11	0	0	1	1	1	11
Rectangle	0	27	0	2	0	0	0	29
Oval	0	27	0	2	0	0	0	29

The proposed example has eight classes. The components of our metric for all classes are summarized in Tab. 2, and non class as global in Tab. 3. Based on these values we have calculated Cclass, CIclass, CDclass, Cglobal, coupling values of the system. It is worth to mention that during the calculation of complexity of inheritance, we should be careful in computing of CIclass. We have to add the complexity of the classes at the same level and only multiply with their parent classes, as shown in the following computation.

Cclass(Colour)=21
Cclass(Shapes)=7
Cclass(Figure1P)=7
Cclass(Square)=29
Cclass(Circle)=29
Cclass(Figure2P)=11
Cclass(Rectangle)=29
Cclass(Oval)=29

Table 3 *Non-class complexity*
Tablica 3. *Složenost ne-razreda*

Non-Class	var+str+obj	Complexity
Cglobal	24	24

CIclass=Shapes*(Figure1P*(Square+Circle+Figure2P*(Rectangle+Oval)))
=7*(7*(29+29+11*(29+29)))
=34104
CDclass=21
Cglobal=24

SMPy=CIclass+CDclass+Cglobal+coupling
=34104+21+24+2^4
=34165.

Calculation is as follows:
1. Complexity of each class was calculated. Attributes, methods, variables, objects, structures, and cohesion were included.
2. Complexity of global structure was calculated. Variables, objects, structures, functions, and the main function were included.
3. Classes were separated as inside inheritance and distinct.
4. Complexity of inheritance was calculated. Super class was multiplied by the summation of the classes, which are derived from it.
5. Complexity of inheritance, complexity of distinct class, complexity of global structure, and coupling were summed to reach the result of SMPy.

5
Empirical validation
Empirijska provjera valjanosti

The empirical validation of the metric is carried out through a case study, comparative study and application of metric on a real project. Case study under consideration is taken into three different languages. We compare our metric with most popular CK metric suite. The case study, comparative study and observations on real projects are demonstrated in section 5.1, 5.2 and in 5.3 respectively.

5.1
Case study
Studija slučaja

For the practical applicability of our metric, we chose a case study. We measure a code, which covers cohesion, coupling, inheritance, polymorphism, attributes, methods, variables, etc. This code, which covers most of possible coding features, is tried in three different languages; C++, Java, and Python. Its UML figure is given in Fig. 1. In other words, we try to develop a system in three different languages and then estimate the complexity of the same system in three different languages. The system is the same, which we considered for the demonstration of metric in section 4. We have already calculated the complexity of the system in Python language in the same section. Now we are estimating the complexity of the same example in C++ and Java. Their class complexity and non Class complexity are given below and in Tab. 4 and 5, respectively. Accordingly,

The metric values for C++
Cclass(Colour)=21
Cclass(Shapes)=7
Cclass(Figure1P)=7
Cclass(Square)=29
Cclass(Circle)=29
Cclass(Figure2P)=11
Cclass(Rectangle)=29
Cclass(Oval)=29

Table 4 *Non-class complexity*
Tablica 4. *Složenost ne-razreda*

Non-Class	var+str+obj	Complexity
Cglobal	32	32

CIclass=Shapes*(Figure1P*(Square+Circle+Figure2P*(Rectangle+Oval)))
=7*(7*(29+29+11*(29+29)))
=34104
CDclass=21
Cglobal=32

Total complexity of the system in
C++=CIclass+CDclass+Cglobal+coupling
=34104+21+32+2^4
=34173.

The metric Values for the Java:
Cclass(Colour)=21
Cclass(Shapes)=7
Cclass(Figure1P)=7
Cclass(Square)=29
Cclass(Circle)=29
Cclass(Figure2P)=11
Cclass(Rectangle)=29
Cclass(Oval)=29

Table 5 *Non-class complexity*
Tablica 5. *Složenost ne-razreda*

Non-Class	var+str+obj	Complexity
Cglobal	71	71

CIclass=Shapes*(Figure1P*(Square+Circle+Figure2P*(Rectangle+Oval)))
=7*(7*(29+29+11*(29+29)))
=34104.
CDclass=21
Cglobal=71
Total complexity of the system in JAVA= CIclass + CDclass + Cglobal + coupling
=34104+21+71+2^4
=34212.

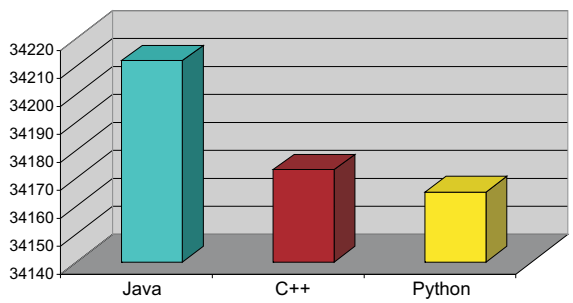


Figure 2 *Comparison between languages*
Slika 2. *Usporedba između jezika*

The data obtained from the above experimentations provide very valuable information regarding the metrics as well as the features of the languages. We can very easily observe that the metric values for classes in all three languages are the same and are: 21, 7, 7, 29, 29, 11, 29 and 29 for classes Colour, Shapes, Figure1P, Square, Circle, Figure2P, Rectangle, and Oval. However the complexity for the whole system in Python, C++ and Java is different and is 34165, 34173 and 34212 respectively (Fig. 2). Here it is important to note that at the class level the complexity values are equal and the differences occur at the main program. This is the reason why the complexity of the same project is different in three different languages. For the whole system, the complexity of the system is the least in python language. It proves the uniqueness of the Python language.

5.2
Comparison with the other metrics
Usporedba s drugim metričnostima

We compare the proposed metric with well-known CK metric suite. We apply all the metrics on the classes of the case study under consideration. The metric values for different classes are summarised in Tab. 6.

Table 6 *Comparison between metrics*
Tablica 6. *Usporedba između metričnosti*

Class / *Metric*	Shapes	Figure1P	Square	Circle	Figure2P	Rectangle	Oval	Colour
WMC(1)	3	2	2	2	2	2	2	2
RFC	3	5	7	7	7	9	9	2
DIT	0	1	2	2	2	3	3	0
NOC	1	3	0	0	2	0	0	0
LCOM	2	2	0	0	2	0	0	0
CBO	0	1	1	1	1	1	1	0
SMpy	7	7	29	29	11	29	29	21

When we compared our metric with CK metric suites, we found that metric values for SMpy are higher than all the CK measures. It is because of the fact that SMpy includes all the parameters, which are responsible for complexity of the systems. However, all these parameters were calculated individually in the CK metric suites. In other words we can say that SMpy is the super set of all the measures proposed by Chidamber et al.[3].

5.3
A real project
Stvarni projekt

The practical usefulness of a new measure cannot be proved without the proper empirical validation which includes the applicability of the metric on real projects. For this purpose, we select an open source project available on the Web. We believe that the open source code is more beneficial for the readers because they can also evaluate the project in the same way as the original author does. The project is a cross-platform set of Python modules designed for writing video games [34]. It includes computer graphics and sound libraries designed to be used with the Python programming language. It is built over the Simple DirectMedia Layer (SDL) library, with the intention of allowing real-time computer game development without the low-level mechanics of the C programming language and its derivatives. This is based on the assumption that the most expensive functions inside games (mainly the graphics part) can be completely abstracted from the game logic in itself, making it possible to use a high-level programming language like Python to structure the game.

We estimate the complexity of each class independently. Classes are coupled through two ways: through inheritances and message calls. The inheritance hierarchies of the classes coupled through the inheritance are shown in Fig. 3. Some classes are independent and therefore not affected due to inheritance and are shown in Fig. 4.

Table 7 *Class complexities*
Tablica 7. *Složenost razreda*

Class	Attributes	Structures	Variables	Objects	MA	AM	Cohesion	Cclass
GameObject	1	1	0	0	1	1	1	3
MapObject	7	27	12	0	1	6	0.1	46,1
Level	2	18	10	0	2	2	1	31
level_zero	1	1	0	0	1	2	0,5	2.5
level_one	3	52	4	1	1	2	0,5	60,5
ImagedObject	2	6	0	0	0	0	0	8
PropTile	0	1	0	0	0	0	0	1
ActorTile	2	3	1	1	0	0	0	7
Sphere	9	9	7	1	2	5	0,4	26,4
Background	1	2	2	0	0	0	0	5
FloorTile	0	1	0	0	0	0	0	1
Cement	3	2	0	0	0	0	0	5
Grass	3	8	1	0	1	1	1	13
Curb	3	2	0	0	1	1	1	6
Void	3	8	3	0	1	1	1	15
Widget	3	24	5	0	4	6	0,6	32,6
Button	10	32	6	0	6	9	1	48,6
DirectionButtons	2	15	4	0	2	2	0,6	22
ViewPort	5	9	0	0	3	5	0,6	14,6
GameObjects	3	7	0	0	3	3	1	11
Clock	11	24	4	0	4	7	0,5	39,5
State	0	1	0	0	0	0	0	1
CyclePath	8	50	6	0	4	8	0,5	64,5

In Tab. 7, the complexity of each class is shown. In the first column the name of the class is given. The metric values for different parameters which affect the complexity of the class i.e. attributes, variables, structures, objects and cohesion are given in column 2 – 7. Cclass is calculated by the equation (2).

It is very easy to observe that the complexity of the class highly depends on its parameter. The highest Cclass is for Button class, which is 48,6 due to complex structure of its methods, number of attributes, and number of variable. The lowest values are for those classes that have simplest structure, for example the class State, Floortile and Proptile.

The complexity of the non classes, defined as Cglobal is due to global variables structures and objects and is computed in Tab. 8. It can be easily observed that global complexity also plays an important role in increasing the overall complexity.

Table 8 *Global complexity*
Tablica 8. *Globalna složenost*

Non-Class	var+str+obj	Complexity
Cglobal	1304	1304

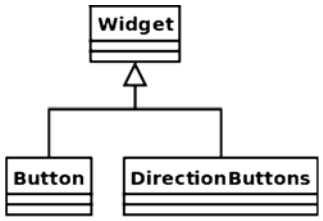


Figure 3 *Inheritance 1*
Slika 3. *Nasljedstvo I*

There are different class hierarchies in this project. The first class hierarchy is shown in Fig. 3. In this hierarchy two classes Button and DirectionButtons are on the same level and inherited from class Widget. Due to the effect of this inheritance the complexity of the class Widget is computed as follows;

Widget(Button + DirectionButtons)
=32,6(48,6 + 22)
=2301,5.

Another class hierarchy which includes 15 classes is shown in Fig. 4. The class Gameobject is at the top of the hierarchy. The complexities of each class under inheritance are given in Tab. 9. The complexity due to inheritance is computed as 275591,1. The demonstration of the calculation for inheritance is given in the following paragraphs.

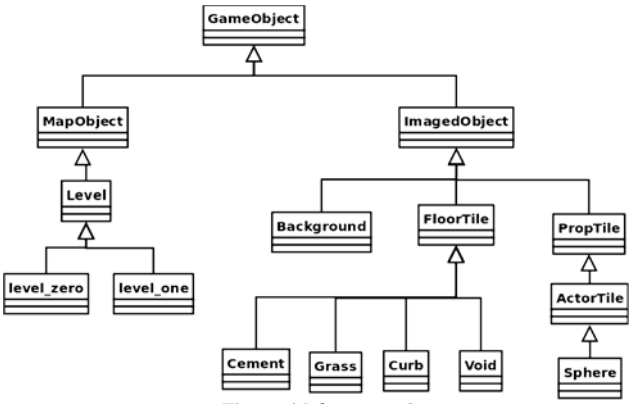


Figure 4 *Inheritance 2*
Slika 4. *Nasljedstvo 2*

GameObject[MapObject(Level(level_zero + level_one))
+ ImagedObject(Background + FloorTile(Cement + Grass
+ Curb + Void) + Proptile(ActorTile(Sphere)))]
=3[46,1(31(2,5+60,5)) + 8(5 + 1(5 + 13 + 6 + 15) +
1(7(26,4)))]
=275591,1.

The different complexities for the project are summarised as follows:

CIclass=277892,6
CDclass=130,6
coupling=0
Cglobal=1304.

Table 9 *Complexity of classes inside inheritance*
Tablica 9. *Složenost klasa unutar nasljedstva*

Class	Complexity
Widget	32.6
Button	48.6
DirectionButtons	22
GameObject	3
MapObject	46.1
ImagedObject	8
Level	31
Background	5
FloorTile	1
PropTile	1
level_zero	2.5
level_one	60.5
Cement	5
Grass	13
Curb	6
Void	15
ActorTile	7
Sphere	26.4

Based on the different complexity values due to different factors the overall complexity of this project is computed as:

SMPy = CIclass + CDclass + coupling + Cglobal
SMPy = 279327,2.

The above computation proves the applicability of SMPy on real life applications. This also proves that not only one factor is responsible for the complexity of the whole code but also there are several factors, which plays the important role in increasing the overall complexity of the code. It is worth mentioning that all these factors are not new but up to now these factors have not been unified for complexity calculation purpose. The complexities for all these factors like inheritance, coupling, methods, are computed independently in the available complexity metric. It is our first attempt to unify all of them in a single metric. In addition, we tried to implement it on the project written in Python. In section 5.1, our experimentation proves that Python is comparatively a better language for the OO software development.

6
Conclusion
Zaključak

There are many metrics for evaluating the quality of codes written in different languages. However, no efforts have been done to propose metrics for Python, which is an important and useful language especially for the software development for the embedded systems. In this present work, we are trying to investigate all the factors, which are responsible for increasing the complexity of code written in Python language. Accordingly, we have proposed a unified metric for this language. Practical applicability of the metric is demonstrated on a case study. We have also validated our work with empirical validation study by applying it on real project. We hope that the present work will attract the attention of the researchers and practitioners who are working in OO domain, especially those using Python.

As a future work, we aim to provide a list of common words (meaningful names) in object oriented languages and in particular programming language, which can be used for

naming the variables and attributes, while writing the code.

7
References
Literatura

[1] Costagliola, G.; Tortora, G. Class points: An approach for the size Estimation of Object-oriented systems. IEEE Transactions on Software Engineering, 31, 1(2005), 52-74.
[2] Misra, S.; Akman, I. Weighted Class Complexity: A Measure of Complexity for Object Oriented Systems. Journal of Information Science and Engineering, 24, (2008), 1689-1708.
[3] Chidamber, S. R.; Kermer, C. F. A Metric Suite for object oriented design. IEEE Transactions Software Engineering, SE-6(1994), 476-493.
[4] http://www.Python.org/about/success/carmanah/
[5] Lutz, M. Learning Python, 4th Edition, Ebook, Safari Books Online, September 2009, Publisher: O'Reilly Media
[6] Fenton, N. E.; Pfleeger, S. L. Software Metrics: A Rigorous and Practical Approach, 2nd Edition Revised ed. Boston: PWS Publishing, 1997.
[7] Bird, S.; Klein, E.; Loper, E. Natural Language Processing with Python, 1st Edition , Ebook, Safari Books Online, June 2009, Publisher: O'Reilly Media
[8] Gift, N.; Jones, M. J. Python for Unix and Linux System Administration, 1st Edition Ebook, Safari Books Online, August 2008, Publisher: O'Reilly Media
[9] Model, M. L.; Tisdall, J. Bioinformatics Programming Using Python, 1st Edition, Ebook, Safari Books Online, July 2009, Publisher: O'Reilly Media
[10] Python Programming Language, cited 04.10.2009. Available from: http://www.Python.org/
[11] Lutz, M. Programming Python, 3rd Edition Ebook, Safari Books Online, August 2006, Publisher: O'Reilly Media
[12] Misra, S.; Akman, I. Unified Complexity Metric: A measure of Complexity, Proc. of National Academy of Sciences Section A. (2010) In press.
[13] Basci, D.; Misra, S. Data Complexity Metrics for Web-Services, Advances in Electrical and Computer Engineering, 9, 2(2009), pp. 9-15.
[14] Basci, D.; Misra, S. Measuring and Evaluating a Design Complexity Metric for XML Schema Documents, Journal of Information Science and Engineering, 25 (2009), pp. 1415-1425.
[15] Wang, Y.; Shao, J. A New Measure of Software Complexity Based On Cognitive Weights. Can. J. Elec. Comput. Engg., (2003), 69-74.
[16] SciPy.in 2009, cited 16.10.2009. Available from: http://scipy.in/
[17] Rupy 2009, cited 10.11.2009. Available from: http://rupy.eu/
[18] FrontRangePythoneersUc09, cited 05.10.2009. Available from: http://wiki.python.org/moin/FrontRangePythoneersUc09
[19] Confoo.Ca Web Techno Conference, cited 14.11.2009. Available from: http://www.confoo.ca/en
[20] Kushwaha, D. S.; Misra, A. K. Improved Cognitive Information Complexity Measure: A Metric That Establishes Program Comprehension Effort. SIGSOFT Software Engineering Notes, 31, 5(Sep. 2006), 1-7.
[21] DMH2000 C/Java/Python/Ruby, cited 15.10.2009. Available from: http://www.dmh2000.com/cjpr/
[22] Neuroscience – Brain vs. Computer, cited 17.10.2009. Available from: http://faculty.washington.edu/chudler/bvc.html
[23] Computer vs. The Brain, cited 17.10.2009. Available from: http://library.thinkquest.org/C001501/the_saga/sim.htm
[24] Basci, D.; Misra, S. Entropy metric for XML DTD documents. SIGSOFT Softw. Eng. Notes, 33, 4(Jul. 2008), 1-6.
[25] Python Code Complexity Metrics And Tools available from: http://agiletesting.blogspot.com/2008/03/Python-code-complexity-metrics-and.html
[26] Pythius Homepage, cited 03.10.2009. Available from: http://pythius.sourceforge.net/
[27] Python Dependency Graphs, cited 08.11.2009. Available from: http://furius.ca/snakefood/
[28] Measuring Cyclomatic Complexity of Python Code available at: http://www.traceback.org/2008/03/31/measuring-cyclomatic-complexity-of-Python-code/
[29] PyMetrics, cited 21.09.2009. Available from: http://sourceforge.net/projects/pymetrics/
[30] Yusuke Shinyama, cited 06.10.2009. Available from: http://www.unixuser.org/~euske/python/index.html
[31] Andersson, M.; Vestergren, P. Object-Oriented Design Quality Metrics, Uppsala Master's Theses in Computer Science 276, 2004-06-07, ISSN 1100-1836.
[32] Babsiya, J.; Davis, C. G. A hierarchical model for object oriented design quality assessment, IEEE Transactions on Software Engineering, 28, 1(2002), pp. 4-17.
[33] Misra, S.; Ferid, C. A Software Metric for Python Language, Proc. of ICSSA 2010, (2010).
[34] http://jtauber.com/pyso/
[35] Dufour, B.; Driesen, K.; Hendren, L.; Verbrugge, C. Dynamic metrics for java. SIGPLAN Notics, 38, 11(2003), 149-168.
[36] Mäkelä, S.; Leppänen, V. A software metric for coherence of class roles in Java programs. In Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java(PPPJ '07), 272(2007), 51-60.

Appendix
Dodatak

A. Code

```
//Shapes program written in C++ to illustrate the usability of the proposed metric
#include <iostream>
#include <string>
using namespace std;
// Colour
class Colour{
    void stars(int limit);
public:
    static char c;
    void getColour();
};
void Colour::getColour(){
    if (c=='s')
        cout<<"Yellow"<<endl;
    else if (c=='c')
        cout<<"Violet"<<endl;
    else if (c=='r')
        cout<<"Red"<<endl;
    else if (c=='o')
        cout<<"Orange"<<endl;
    else
        cout<<"White"<<endl;
        stars(5);
}
void Colour::stars(int limit){
    int outer_loop, inner_loop;
    for (outer_loop=limit; outer_loop>0; outer_loop--){
        for (inner_loop=1; inner_loop<=outer_loop;
inner_loop++)
            printf("*");
        printf("\n");
    }
```

```
}
// -----------
char Colour::c;

class Shapes {
public:
    Shapes(int px, int py):x(px),y(py) {}
    int x, y; //position
    virtual string type() = 0;
    virtual void info() {
        cout << endl << "figure: " << type() << endl;
        cout << "position: x=" << x << ", y=" << y <<
endl;
    }
};
class Figure1P : public Shapes {
public:
    Figure1P(int px, int py, int r):p1(r),Shapes(px, py) {}
    int p1;
    virtual void info() {
        Shapes::info();
        cout << "property 1: p=" << p1 << endl;
    }
};
class Square : public Figure1P {
public:
    Colour *its_colour;
    Square(int px, int py, int r):Figure1P(px, py, r) {}
    virtual string type() {
        Colour::c='s';
        its_colour->getColour();
        return "square";
    }
};
class Circle : public Figure1P {
public:
    Colour *its_colour;
    Circle(int px, int py, int r):Figure1P(px, py, r) {}
    virtual string type() {
        Colour::c='c';
        its_colour->getColour();
        return "circle";
    }
};
class Figure2P : public Figure1P {
public:
    Figure2P(int px, int py, int w, int
h):p2(h),Figure1P(px, py, w) {}
    int p2;
    virtual void info() {
        Figure1P::info();
        cout << "property 2: p=" << p2 << endl;
    }
};

class Rectangle : public Figure2P {
public:
    Colour *its_colour;
    Rectangle(int px, int py, int w, int h):Figure2P(px, py,
w, h) {}
    virtual string type() {
        Colour::c='r';
        its_colour->getColour();
        return "rectangle";
    }
};

class Oval : public Figure2P {
public:
    Colour *its_colour;
    Oval(int px, int py, int w, int h):Figure2P(px, py, w, h)
{}
    virtual string type() {
        Colour::c='o';
        its_colour->getColour();
        return "oval";
    }
};
// Freeing memory
void freeRAM(Shapes *objs[], int i){
    delete objs[i];
}
// --------------
int main(void) {
    Shapes **objs = new Shapes*[5];
    // creating objects
    objs[0] = new Circle(7, 6, 55);
    objs[1] = new Rectangle(12, 54, 21, 14);
    objs[2] = new Square(19, 32, 10);
    objs[3] = new Oval(43, 10, 4, 3);
    objs[4] = new Square(3, 41, 3);
    bool flag=false;
    do {
        cout << endl << "We have 5 objects with
numbers 0..4" << endl;
        cout << "Enter object number to view
information about it " << endl;
        cout << "Enter any other number to quit " <<
endl;
        char onum; // in fact, this is a character, not a
number
        // this allows user to enter letter and quit...
        cin >> onum;
        // flag -- user have entered number 0..4
        flag = ((onum >= '0')&&(onum <= '4'));
        if (flag)
            objs[onum-'0']->info();
    }while(flag);
    for(int i=0;i<5;i++)
        freeRAM(objs,i);
    delete [] objs;
    return(0);
}
```

B:

Table 10 *Values/weights of basic control structures W(BCS) (given in Tab. 1) for all classes of the case study*
Tablica 10. *Vrijednosti/težine osnovnih kontrolnih struktura W(BCS) (datih u Tab. 1) za sve razrede analiziranog slučaja.*

Class	Values of all categories except condition and loop	Condition	Loop	Total
Colour	2	4*2	3*3	19
Shapes	6	0	0	6
Figure1P	8	0	0	8
Square	27	0	0	27
Circle	27	0	0	27
Figure2P	11	0	0	11
Rectangle	27	0	0	27
Oval	27	0	0	27

Classes like *Square*, *Circle*, *Rectangle*, *Oval* have higher structural statement complexity due to function calls. Each function call has a weight of 2 BCS, but additionally, the complexity weight of the called function should be included, too.

For example:

Figure1P has only 1 sequence structure. However, it calls *Shapes*' *constructor* and *info* method. Due to each call (2+2) is added. Total makes up 5. Moreover, the *constructor* has a complexity weight of 1 and the *info* method has a complexity of 2. Thus, (5+1+2) makes up 8 which is *Figure1P*'s class complexity.

Similar example can be observed also with the *Rectangle* class. *Rectangle* calls *Figure2P*'s *constructor* and also *Colour*'s *getColour* method. Even though each function call is weighted as 2, after adding Figure2P's constructor and *Colour*'s *getColour* structural complexities, total class complexity of *Rectangle* becomes higher.

Authors' addresses
Adrese autora

Prof. (Dr.) Sanjay Misra
Department of Computer Engineering
Faculty of Engineering
Atilim Uíversity, Ankara
Turkey

Ferid Cafer
Software Engineer
Servus Bilgisyar, Ankara
Turkey